一种基于 IBPSO 的 SDN 控制器放置优化方案 *

王 潇，杨金民

(湖南大学 信息科学与工程学院，长沙 410082)

摘 要：软件定义网络（software defined network，SDN）通过控制器实现网络的灵活管控，因此控制器放置对网络整体性能至关重要。提出了改进离散粒子群优化（improved binary particle swarm optimization，IBPSO）算法用来解决控制器放置问题。该算法基于粒子群的全局最优和单个粒子的个体最优来决定粒子当前取值概率，消除粒子当前值对下一步迭代的影响，从而加快收敛速度，找到更优的最终结果。仿真结果表明，与离散粒子群优化（binary particle swarm optimization，BPSO）算法相比，由该算法得出的控制器放置方案在实现控制器负载均衡的同时，还可以显著降低控制器的数量。

关键词：软件定义网络；控制器放置；负载均衡；粒子群算法

中图分类号：TP393.0 doi: 10.3969/j.issn.1001-3695.2018.03.0225

Optimization scheme for SDN controllers' placement based on IBPSO

Wang Xiao, Yang Jinmin

(*College of Computer Science & Electronic Engineering Hunan University*, *Changsha* 410082, *China*)

Abstract: The software defined network (SDN) realizes the flexible control of the network through the controller, so the placement of the controller is critical to the overall performance of the network. This paper proposed an improved improved binary particle swarm optimization (IBPSO) algorithm to solve the controller placement problem. IBPSO algorithm determines the current value of particles based on both the global optimal and the individual optimal, thus weakening the influence of particle current value on the next iteration, and accelerating the convergence, which leaded to a better final result. The simulation results show that compared with the BPSO algorithm, the controller placement scheme obtained by this algorithm can significantly reduce the number of controllers while realizing the load balancing of the controller.

Key words: software defined network; controllers' placement; load balancing; particle swarm algorithm

0 引言

传统网络难以满足云计算、大数据，以及相关业务提出的灵活的资源需求。传统网络设备的封闭性，使得在真实环境中规模部署新协议时通常会引发出一系列问题。因此，软件定义网络应运而生。软件定义网络(software defined network，SDN)[1]是一种新型的网络架构，它的核心理念是将网络的控制层面和数据转发层面进行分离，从而实现对网络的集中化控制。

SDN 将原来封闭在交换机或路由器中的控制平面分离出来，形成一个网络部件，称之为 SDN 控制器。SDN 通过控制器来实现对数据转发设备的统一控制。在单控制器 SDN 网络中如 NOX[2]、Beacon[3]、Floodlight[4]，随着网络规模的不断增加，控制器的处理能力将会成为系统的瓶颈。因此，研究人员提出了多控制器 SDN 网络，如 Onix[5]、Hyperflow[6]、Kandoo[7]等。多控制器 SDN 网络通过在网络中部署多台控制器从而实现逻辑上的集中式控制。

对于多控制器 SDN 网络来说，一个关键问题就是控制器放置问题（controller placement problem，CPP）。CCP 问题最早是由 Heller 等人[8]提出，其控制器位置是通过最小化控制器与交换机之间的平均时延和最大时延来确定的。文献[9]提出了一种基于密度的控制器放置算法，以解决控制器部署在延迟、容错和控制器数量方面的要求，但是未将控制器负载考虑在内。一种基于流管理的控制器放置策略，包括将业务负载迁移到冗余链路[10]、动态的添加或删除控制器、关闭相关连接[11]被提出用来节省开销。但是流管理会增加额外的成本，同时网络性能与节能之间的权衡也需要深入评估。

现有针对 CCP 问题的研究主要分为四类[12]，即减少交换机与控制器之间时延[8,13]、提高可靠性和弹性[14]、减低部署成本和资源消耗[15]、多目标方法[16]。但是在真实环境中，控制器的处理能力对网络的性能至关重要。如果控制器的负载过重，

收稿日期：2018-03-18；**修回日期**：2018-05-16 **基金项目**：国家自然科学基金资助项目（61272401）；湖南省科技计划重点项目（2013GK2003）

作者简介：王潇（1994-），女，黑龙江大兴安岭人，硕士研究生，主要研究方向为软件定义网络；杨金民（1967-），男，湖南宁乡人，教授，博士，主要研究方向为软件工程、系统可靠性、数据挖掘、大数据（rj_jmyang@hnu.edu.cn）.

控制器将不能对流请求进行及时的处理，从而影响网络的性能；如果控制器负载过轻，控制器就未得到充分的利用，从而造成资源浪费。因此，合理的控制器放置方案能促进控制器的高效利用，使控制平面负载均衡。

本文将控制器的负载作为优化变量，建立面向 SDN 负载优化的数学模型，得出启发式的优化求解方案。该方案使用改进的离散粒子群优化算法即 IBPSO 算法来得出控制器的放置方案。该方案首先将控制器放置问题模型化为粒子群中的粒子位置优化问题；然后将控制器处理能力作为约束条件，将控制器负载均衡度作为适应度函数来评价部署方案。每个粒子通过迭代不断向历史最佳位置和全局最佳位置优化，最终得到近似最优的 SDN 控制器放置方案。

1 SDN 控制器放置问题建模

SDN 控制器放置问题就是在一个给定的网络拓扑中，确定网络中的控制器与哪些交换机相连，能够使整个网络中的负载分布更均衡。SDN 的网络拓扑可模型化为一个图 G(V,E)，其中 V 是图 G 的结点集合，表示网络中的交换机集合，$V=\{v_1,v_2,...v_n\}$，|V|=n；E 为图 G 的边集合，表示交换机之间直接相连的的物理链路的集合；{C}表示控制器集合，|C|= m。假定在向网络 G 中部署控制器时，可以在网络中任何放置交换机的节点处放置控制器。交换机与控制器之间的关系用 n×m 的分布矩阵 X=（x_{ij}）来表示。x_{ij} 表示交换机 V_i 与控制器 C_j 之间的连接关系，如果 x_{ij} 为 1，则表示控制器 C_j 是交换机 V_i 的主控制器，否则 x_{ij} 取值为 0。

在 SDN 网络中，控制器的负载来源于交换机的 packet-in 消息，以及控制器之间保持视图一致性的交换信息。其中，源于交换机的 packet-in 消息占控制器负载的主要部分，所以本文使用交换机向控制器发送的packet-in 消息数作为控制器负载的衡量指标。控制器 C_j 的负载用 L_j 表示，f_i 表示交换机的流请求数，控制器 C_j 的负载 L_j 可表示为

$$L_j=\sum_{V_i\in V}x_{ij}f_i \tag{1}$$

控制器 C_j 的最大处理能力用 c_j 表示，于是控制器 C_j 的利用率 $u_j= L_j/c_j$。SDN 网络中所有控制器的平均利用率为$\bar{u}$ 。

$$\bar{u}=\frac{1}{m}\sum_{C_j\in C}u_j \tag{2}$$

控制器的负载均衡度为 Q。

$$Q=\frac{1}{m}\sum_{j=1}^{m}(u_j-\bar{u})^2 \tag{3}$$

SDN 控制器部署的优化模型为 min Q，Q 的取值范围为（0，1）。

考虑到控制器的处理能力，所以每个控制器所处理的请求数量不能大于 c_j，即利用率：

$$u_j\le 1 \tag{4}$$

根据 OpenFlow[17]协议，每一个交换机只能由一个主控制器控制，有约束条件：

$$\forall i,\ \sum_{C_j\in C}x_{ij}=1$$

$$\forall i,j,x_{ij}\in(0,1) \tag{5}$$

由式（5）可知，控制器负载均衡模型求解问题是一个 0-1 整数规划问题。针对 0-1 整数规划问题，常采用启发式算法来进行求解。因此，可以采用改进的离散粒子群优化算法来求解该模型。

2 改进的粒子群优化算法

BPSO 算法是一种求解离散问题的启发式搜索算法。BPSO 在求解时将问题的解抽象为搜索空间中的粒子，并通过适应度函数来评估粒子的适应值。BPSO 将粒子的每一维的位置向量 x_i 限定为 1 或者 0，速度根据式（6）更新。用速度更新位置时，BPSO 采用 *sigmoid* 函数将速度映射到[0,1]区间，如式（8）（9）所示。如果 V_i 高一些，粒子的位置 x_i 更有可能选 1，V_i 低一点则 x_i 更有可能选 0。其中：*rand*（）是（0,1）之间的随机数；*pbest* 为个体历史最优值；*gbest* 为种群历史最优值；c_1 和 c_2 为学习因子，是非负常数，通常取值为 2；ω 表示惯性因子。

$$V_i=\omega\times V_i+c_1\times rand()\times(pbest_i-x_i)+c_2\times rand()\times(gbest_i-x_i) \tag{6}$$

$$S\left(v_i^{t+1}\right)=\frac{1}{1+e^{-v_i^{t+1}}} \tag{7}$$

$$x_i^{t+1}=\begin{cases}1, rand()\le S(v_i^{t+1})\\0, otherwise\end{cases} \tag{8}$$

BPSO 算法通过 *sigmoid* 函数将 V_i 转变为 x_i 取 0 或者取 1 的概率，未能充分利用粒子群算法的性能[18]。考虑到在现实社会中，人在做决定时往往会依赖于个人以往的历史经验以及群体的历史经验，因此，本文提出了 IBPSO 算法。该算法根据控制器放置问题的特点，重新定义了粒子及粒子位置更新过程。每个粒子代表一种控制器部署方案，在对粒子位置更新时根据粒子个体历史最优解 *pbest* 和种群历史最优解 *gbest* 由贝叶斯公式来决定粒子位置取值的概率。

2.1 粒子位置定义

假设 SDN 网络中有 m 个控制器，交换机的数量为 n。在此初始化一群粒子，设初始化粒子的个数为 N，每个粒子代表一种控制器部署方案。每个粒子都可以用一个 n 维向量 X_i 表示 。其中 $X_i=[x_{i1},x_{i2}....x_{in}]$, n 为交换机的个数, x_{il} 表示在第 i 种可行方案中，编号为 l 的交换机连接到的控制器编号。若交换机 j 与交换机 k 连接到同一个控制器，则有 $x_{ij}=x_{ik}$ 。在第 i 中可行方案中交换机与控制器的连接关系可以用一个（0,1）位置矩阵 CC_i 来表示：

$$CC_i = \begin{bmatrix} c_{11}{}^i & c_{12}{}^i & \cdots & c_{1k}{}^i & \cdots & c_{1m}{}^i \\ c_{21}{}^i & c_{22}{}^i & \cdots & c_{2k}{}^i & \cdots & c_{2m}{}^i \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ c_{l1}{}^i & c_{l2}{}^i & \cdots & c_{lk}{}^i & \cdots & c_{lm}{}^i \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ c_{n1}{}^i & c_{n2}{}^i & \cdots & c_{nk}{}^i & \cdots & c_{nm}{}^i \end{bmatrix}$$

在位置矩阵 CC_i 中如果交换机 l 连接到控制器 k，则在位置矩阵 CC_i 中元素 $c_{lk}{}^i=1$，否则 $c_{lk}{}^i=0$。根据式（5）可知，

$\sum_{k=1}^{m} c_{lk}^i = 1, \forall l \in \{1,2,3...,n\}$。

2.2 粒子位置更新过程

假设交换机 i 在第 k+1 次位置更新时连接到控制器 h 的概率为 $P(X_{ih}{}^{k+1}=1)$, 反之为 $P(X_{ih}{}^{k+1}=0)$。在完全随机状态下 $P(X_{ih}{}^{k+1}=1)=P(X_{ih}{}^{k+1}=0)=0.5$，

假定在求解过程中，*pbest* 和 *gbest* 对最佳值的判定是相互独立的。设粒子个体最优解的信任度为 *Pp*, 全局最优解的信任度为 *Pg*。*pbest* 和 *gbest* 是迭代过程中的历史最佳值，因此它们发现最优值的概率应该超过平均值，即 $Pp>0.5$，$Pg>0.5$，且 $Pg<Pp<1$。

引入 α 和 β 两个变量，交换机 x 在第 k+1 次位置更新时连接到控制器 h 的概率由贝叶斯公式可得

$$P(X_{ih}^{k+1}=1 \mid pbest_{ih}^{k}=1, gbest_{ih}^{k}=1) = \frac{P_p \times P_g}{P_p \times P_g + (1-P_p)\times(1-P_g)} = \alpha \tag{9}$$

$$P(X_{ih}^{k+1}=1 \mid pbest_{ih}^{k}=0, gbest_{ih}^{k}=0) = \frac{(1-P_p)\times(1-P_g)}{P_p \times P_g + (1-P_p)\times(1-P_g)} = 1\text{-}\alpha \tag{10}$$

$$P(X_{ih}^{k+1}=1 \mid pbest_{ih}^{k}=1, gbest_{ih}^{k}=0) = \frac{P_p \times (1-P_g)}{P_p \times P_g + (1-P_p)\times(1-P_g)} = \beta \tag{11}$$

$$P(X_{ih}^{k+1}=1 \mid pbest_{ih}^{k}=0, gbest_{ih}^{k}=1) = \frac{(1-P_p)\times P_g}{P_p \times P_g + (1-P_p)\times(1-P_g)} = 1\text{-}\beta \tag{12}$$

当 *Pp* 取值为 0.7，Pg 为 0.8，此时 $\alpha=0.9$，$\beta=0.7$，IBPSO 算法可取得最佳性能[19]。因此，本文令 $Pp=0.7$，$Pg=0.8$。位置更新过程伪代码如下：

算法 1 位置更新过程

```
r=rand ();
if pbest^t=1 and gbest^t=1 then
    if r<α then x^t=1
    else x^t=0
else
if pbest^t=0 and gbest^t=0 then
    if r<1-α then x^t=1
    else x^t=0
else
if pbest^t=1 and gbest^t=0 then
    if r<β then x^t=1
    else x^t=0
else
    if r<1-β then x^t=1
else
    x^t=0
```

2.3 适应度函数定义

适应度函数用来评估粒子取值的好坏。本文将控制器的控制器负载均衡度作为适应度函数：

$$Q = \frac{1}{m}\sum_{j=1}^{m}(u_j - \bar{u})^2 \tag{13}$$

IBPSO 算法具体流程如下：

(a) 输入网络拓扑 G（V,E），设置控制器最大负载阈值 $C^{th}=(c_1,c_2,...c_m)$，请求流向量 F=（$f_1,f_2,...f_n$）。设置最大迭代次数 T，设置种群大小 PopSize。

(b) 初始化粒子的位置 $X_i=\{x_1,x_2,....x_n\}$。

(c) 初始化每个粒子的个体最优值。

(d) 计算全局最优解。

(e) 根据位置更新公式更新粒子的位置，得到 X_{i+1}。

(f) 判断粒子是否满足约束条件（d）（e）如果是则进入步骤(g)，否则转向步骤(b)。

(g) 根据式（13）计算新的状态的粒子的适应度。

(h) 更新每个粒子的个体最优解。

(i) 更新粒子的全局最优解。

(j) 如果经过多次迭代全局最优解不再发生改变或者迭代次数大于 T，算法停止，输出全局最优解；否则算法回到步骤(e)，进行下一次迭代。

2.4 算法分析

IBPSO 算法通过条件概率得到粒子取值的概率，搜索效率更高。而 BPSO 算法要先计算粒子的速度，通过 *sigmoid* 函数将 V_i 转变为 x_i 取 0 或者取 1 的概率。

IBPSO 算法在位置更新的过程中根据 *pbest* 和 *gbest* 来决定位置向量 x_i 的取值概率。在 BPSO 算法中，x_i 的值受到上一次迭代结果的影响，随着迭代次数的增加，BPSO 算法的随机性增强。

IBPSO 算法的收敛性受到 α 和 β 两个变量取值的影响。其中 α 取值表示当个体最优解和种群最优解相同时，个体最优解和种群最优解对 x_i 取值的影响，而 β 表示当个体最优解和种群最解不同时，个体最优解和种群最优解对 x_i 取值的影响。当 $0.5<\beta<\alpha<1$ 时 IBPSO 算法的收敛性最佳，而 $\alpha=\beta=0.5$ 时，IBPSO 算法就变成了一种随机搜索算法，算法的收敛性较差。

本文根据 IBPSO 算法获得 SDN 控制器放置方案，从而使整个网络中的控制器处于负载均衡的状态。因此，通过仿真实验来验证本文算法的有效性。

3　实验仿真与结果分析

本文选用的仿真平台是 MATLAB 2010，并对 MATLAB 中的 biography 做修改。搭建出可以模拟真实网络环境的网络仿真平台。在进行仿真实验时采用的是 Salama 模型随机生成网络拓扑。本文假设拓扑中的每个节点都可以部署交换机，控制器可以部署在每个交换机的位置。本文中预先设定实验中所使用的控制器每分钟能够处理 2 000 条 packet-in 消息。为了比较本文算法的性能，从控制器数量、网络中控制器的负载均衡度以及算法的收敛性三个方面将 IBPSO 与 BPSO 算法作比较。

3.1　控制器数量

图 1 显示的是在不同网络规模下，两种算法在控制器部署中对控制器数量的要求。首先先预定一个较小的值 k，并逐渐增加 k 的大小，直到找到满足各种约束的部署方案。此时，刚好没有控制器出现超载情况。由图 1 可知，采用 IBPSO 求解的控制器个数处于一个较低的水平。对于网络规模为 50 km（fraction of topologies=0.5）的网络来说，IBPSO 得到的部署方案是需要 4 个控制器；对于网络规模达到 90 km（fraction of topologies=0.9）的网络来说，IBPSO 需要部署 8 个控制器。相同条件下，BPSO 算法分别需要部署 6 和 10 个控制器。

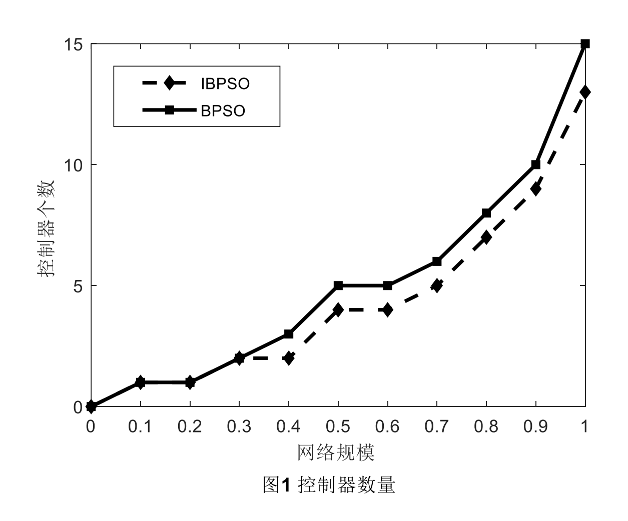


图1 控制器数量

3.2　负载均衡度

本文使用负载均衡度来度量网络中各控制器负载的差异程度。负载均衡度的大小表示网络中控制器负载差异的大小，当负载均衡度值大小，表明了控制器间负载不均衡现象严重程度。图 2 表示，在相同的网络环境下，本文部署相同数量的控制器，两种算法下的控制器负载均衡度的情况。实验结果表明，使用 IBPSO 算法得到的控制器放置方案在负载均衡上优于由 BPSO 算法得到的放置方案。

3.3　收敛性

在相同的网络拓扑下，通过设置相同的控制器个数来比较 IBPSO 和 BPSO 算法的性能。经过 40 次的迭代，IBPSO 得到了最佳适应值，而 BPSO 则需要经过 50 次的迭代才能够达到最佳适应值。在相同的迭代次数下，IBPSO 取得的适应值要优于 BPSO 取得的适应值。由图 3 可知，IBPSO 算法较之 BPSO 算法具有更好的收敛性。

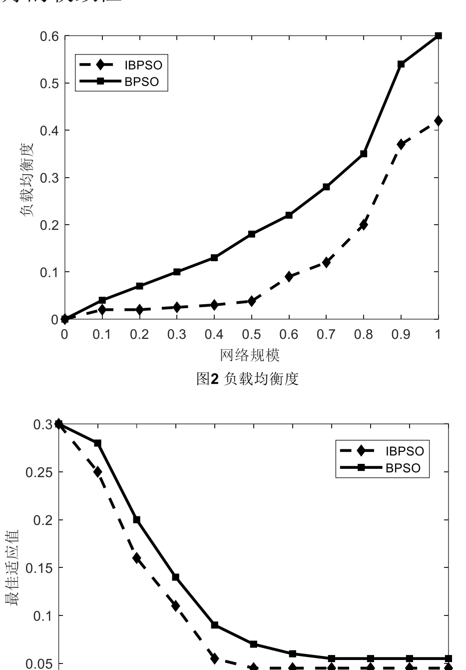


图2 负载均衡度

图3 收敛性

4　结束语

本文将控制器的负载作为优化变量，提出 SDN 控制器部署优化模型，并将改进的二进制粒子群优化算法用于求解优化模型。IBPSO 算法通过条件概率直接决定粒子粒子取值的概率，解决了 BPSO 求解过程中参考上一次迭代中 x_i 的值而陷于局部最优的问题，具有更好的收敛性。同时仿真实验结果表明，使用 IBPSO 算法能够有效的减少控制器部署个数，实现控制器负载的均匀分配。

参考文献：

[1] Kreutz D, Ramos F M V, Esteves Verissimo P, *et al*. Software-defined networking: a comprehensive survey [J]. Proceedings of the IEEE, 2014, 103 (1): 10-13.

[2] Gude N, Koponen T, Pettit J, *et al*. NOX: towards an operating system for networks [J]. ACM Sigcomm Computer Communication Review, 2008, 38 (3): 105-110.

[3] Erickson D. The beacon openflow controller [C]// Proc of ACM SIGCOMM HOTSDN. New York: ACM Press, 2013: 13-18.

[4] Floodlight [EB/OL]. [2013-12-29]. http://www. projectfloo-dlight. org.

[5] Koponen T, Casado M, Gude N, *et al*. Onix: a distributed control platform for large-scale production networks [C]// Proc of the 9th USENIX Conference Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2010: 351-364.

[6] Tootoonchian A, Ganjali Y. HyperFlow: a distributed control plane for OpenFlow [C]// Proc of Internet Network Management Conference on Research on Enterprise Networking. Berkeley: USENIX Association, 2010.

[7] Yeganeh S H, Ganjali Y. Kandoo: a framework for efficient and scalable offloading of control applications [C]// Proc of the 1st Workshop on Hot Topics in Software Defined Networks. New York: ACM Press, 2012: 19-24.

[8] Heller B, Sherwood R, Mckeown N. The controller placement problem [C]// Proc of ACM SIGCOMM HOTSDN. New York: ACM Press, 2012: 19-24.

[9] Liao Jianxin, Sun Haifeng, Wang Jingyu, *et al*. Density cluster based approach for controller placement problem in large-scale software defined networkings [J]. Computer Networks, 2017, 112: 24-35.

[10] Muller L F, Oliveira R R, Luizelli M C, *et al*. Survivor: an enhanced controller placement strategy for improving SDN survivability [C]// Proc of Global Communications Confere-nce. [S. l.] : IEEE Press, 2014: 1909-1915.

[11] Jimenez Y, Cervello-Pastor C, Garcia A J. On the controller placement for designing a distributed SDN control layer [C]// Proc of Networking Conference. [S. l.] : IEEE Press, 2014: 1-9.

[12] Wang Guodong, Zhao Yanxiao, Huang Jun, *et al*. The controller placement problem in software defined networking: a survey [J]. IEEE Network, 2017, 31 (5): 21-27.

[13] Han Lin, Li Zhiyang, Liu Weijiang, *et al*. Minimum control latency of SDN controller placement [C]// Proc of Trustcom//Bigdatase//Ispa. [S. l.] : IEEE Press, 2017: 2175-2180.

[14] Tanha M, Sajjadi D, Pan J. Enduring node failures through resilient controller placement for software defined networks [C]// Proc of Global Communications Conference. [S. l.] : IEEE Press, 2017.

[15] Sallahi A, St-Hilaire M. Expansion model for the controller placement problem in software defined networks [J]. IEEE Communications Letters, 2016, PP (99): 1.

[16] Zhang Bang, Wang Xingwei, Ma Lianbo, *et al*. Optimal controller placement problem in Internet-oriented software defined network [C]// Proc of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. 2017: 481-488.

[17] Mckeown N, Anderson T, Balakrishnan H, *et al*. OpenFlow: enabling innovation in campus networks [C]// Proc of ACM Sigcomm Computer Communication Review. New York: ACM Press, 2008: 69-74.

[18] 刘建华，杨荣华，孙水华. 离散二进制粒子群算法分析 [J]. 南京大学学报：自然科学版, 2011 (5): 504-514. (Liu Jianhua, Yang Ronghua, Liu Shuihua. The analysis of binary particle swarm optimization [J]. Journal of Nanjing University: Nat Sci Ed, 2011 (5): 504-514.)

[19] 徐义春，肖人彬. 一种改进的二进制粒子群算法 [J]. 模式识别与人工智能, 2007, 20 (6): 788-793. (Xu Yichun, Xiao Renbin. An improved binary swarm optimizer [J]. Pattern Recognition and Artificial Intelligence, 2007, 20 (6): 788-793.)